• Connecting Django to a database (SQLite or MySQL).

Connecting Django to a database is a crucial step in setting up your Django project. By default, Django comes configured to use SQLite, but you can also connect it to other databases like MySQL. Here’s how to do both:

**Connecting to SQLite (Default)**

SQLite is the default database in Django, and it's suitable for development and testing.

1. **Check Default Configuration** By default, your settings.py file in the Django project directory includes the configuration for SQLite:

* python

# myproject/settings.py

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.sqlite3',

'NAME': BASE\_DIR / 'db.sqlite3',

}

}

1. **No Additional Steps Required** SQLite requires no additional setup. You can start using the database right away.

**Connecting to MySQL**

To use MySQL, you'll need to install the necessary database driver and configure your Django settings.

1. **Install MySQL Client** You need to install mysqlclient or mysql-connector-python to allow Django to interact with MySQL.

pip install mysqlclient

or

bash

pip install mysql-connector-python

1. **Configure Database Settings** Update the DATABASES setting in your settings.py file to use MySQL.

* python

# myproject/settings.py

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.mysql',

'NAME': 'your\_database\_name',

'USER': 'your\_database\_user',

'PASSWORD': 'your\_database\_password',

'HOST': 'your\_database\_host', # Usually 'localhost' or '127.0.0.1'

'PORT': 'your\_database\_port', # Default MySQL port is '3306'

}

}

1. **Create the Database** Ensure you have created the database in MySQL that matches the NAME specified in settings.py.

* sql

CREATE DATABASE your\_database\_name;

1. **Apply Migrations** Once your settings are configured, apply Django migrations to set up the database schema.

python manage.py migrate

• Using the Django ORM for database queries.

The Django ORM (Object-Relational Mapping) provides a powerful and intuitive way to interact with your database using Python code. Instead of writing raw SQL queries, you can use Django's ORM to perform database operations.

**Defining Models**

First, define your models in models.py. These models represent the structure of your database tables.

* python

# myapp/models.py

from django.db import models

class Car(models.Model):

make = models.CharField(max\_length=50)

model = models.CharField(max\_length=50)

year = models.IntegerField()

def \_\_str\_\_(self):

return f"{self.make} {self.model} ({self.year})"

**Performing Basic Queries**

Django ORM allows you to perform various database operations easily. Here are some common queries:

**1. Creating Records**

You can create new records in the database using the create() method or by creating an instance and saving it.

* python

# Using create()

Car.objects.create(make="Toyota", model="Corolla", year=2020)

# Creating an instance and saving

car = Car(make="Honda", model="Civic", year=2021)

car.save()

**2. Retrieving Records**

There are several ways to retrieve records using the ORM:

* **All Records**: Retrieve all records from a table.
* python

cars = Car.objects.all()

* **Filter Records**: Filter records based on certain criteria.
* python

toyota\_cars = Car.objects.filter(make="Toyota")

* **Get a Single Record**: Retrieve a single record that matches the criteria.
* python

car = Car.objects.get(id=1)

* **Order Records**: Order records by a specific field.
* python

ordered\_cars = Car.objects.order\_by('year')

**3. Updating Records**

To update a record, first retrieve it, modify the necessary fields, and then save it.

* python

car = Car.objects.get(id=1)

car.year = 2022

car.save()

**4. Deleting Records**

To delete a record, retrieve it and call the delete() method.

* python

car = Car.objects.get(id=1)

car.delete()

**Advanced Queries**

The ORM also supports more advanced queries:

**1. Field Lookups**

Field lookups allow you to filter records using different conditions.

* python

# Filtering records with greater than condition

recent\_cars = Car.objects.filter(year\_\_gt=2018)

# Filtering records containing a substring

corolla\_cars = Car.objects.filter(model\_\_icontains="Corolla")

**2. Aggregation**

You can perform aggregation operations like count, average, sum, etc.

* python

from django.db.models import Avg, Count

# Counting the number of cars

num\_cars = Car.objects.count()

# Calculating the average year

average\_year = Car.objects.aggregate(Avg('year'))